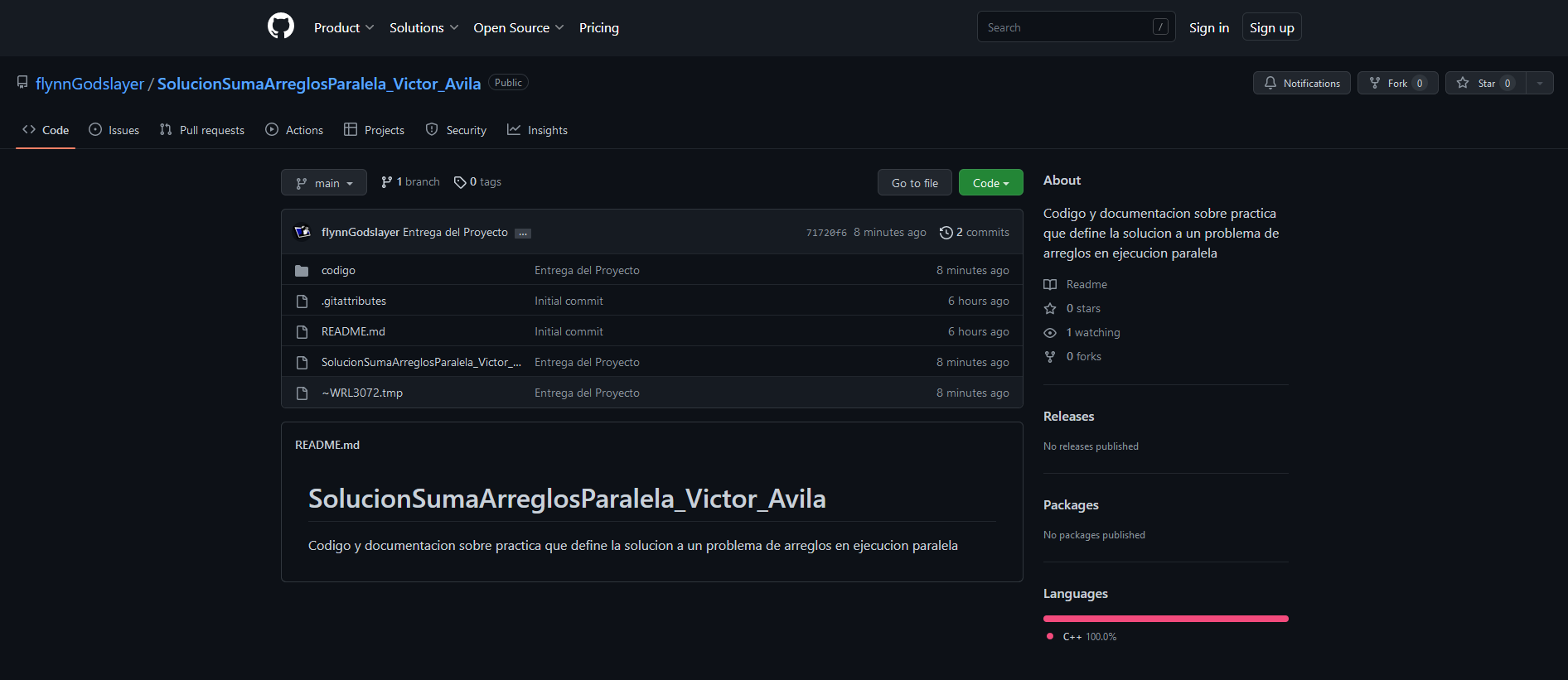
# 

# Tarea 1. Programación de una solución paralela

## Liga del repositorio en GitHub

En el siguiente repositorio se puede consultar la presente documentación y el código del proyecto:

(<https://github.com/flynnGodslayer/SolucionSumaArreglosParalela_Victor_Avila>)



El repositorio se encuentra dividido en 2:

Apenas entrar en el repositorio, se encuentra el presente documento.

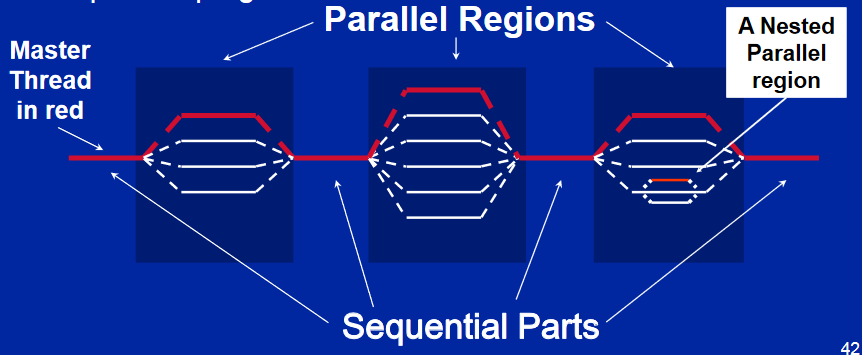


Dentro de la carpeta código se encuentra el proyecto en C++ que soluciona el problema de tarea.



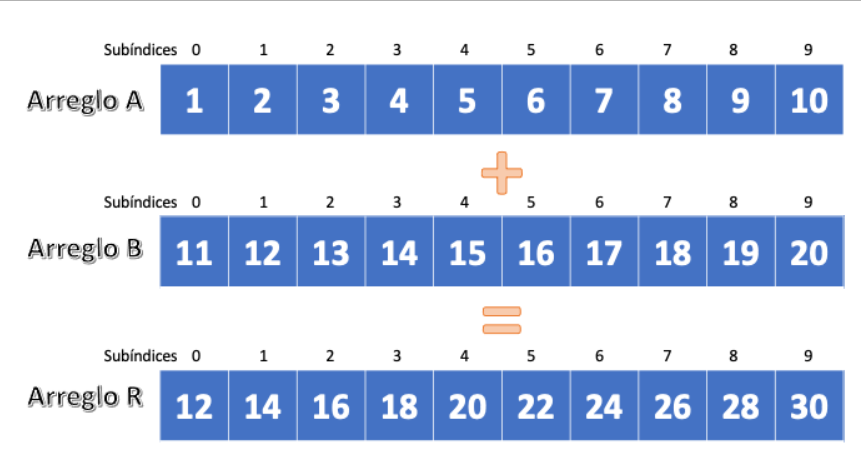
## Introducción

Open MP es un modelo de programación en paralelo que permite llevar a cabo una configuración de ejecución de los programas escritos en C++ por medio de hilos (threads). Hace uso del modelo fork-join en paralelo, donde un hilo maestro (Master thread) ejecuta un equipo de tantos sub-hilos como necesite.



En esta práctica, se desea resolver el siguiente problema:

Piense en una suma de dos arreglos como la mostrada en la siguiente imagen, donde tenemos dos arreglos llamados A y B de 10 elementos cada uno y deseamos crear un tercer arreglo con la sumatoria de cada uno de los elementos en los mismos subíndices.



Esto es bastante sencillo en la programación, pero el tiempo que tomará en calcularse el resultado depende directamente de la cantidad de elementos que tienen los arreglos y la velocidad del procesador en la que se ejecutan las operaciones. Las sumas individuales de los elementos se realizan una por una en orden según los subíndices, entonces ¿Qué pasaría si los arreglos tienen millones de elementos?

## Solución del algoritmo

La respuesta es que la suma de los arreglos tardaría bastante en resolverse. Entonces aquí podríamos aplicar la programación paralela para que las sumas de algunos elementos de los arreglos se realicen en un thread, otra parte las controle otro thread y así sucesivamente, esto aprovechando que las sumas son independientes entre ellas.

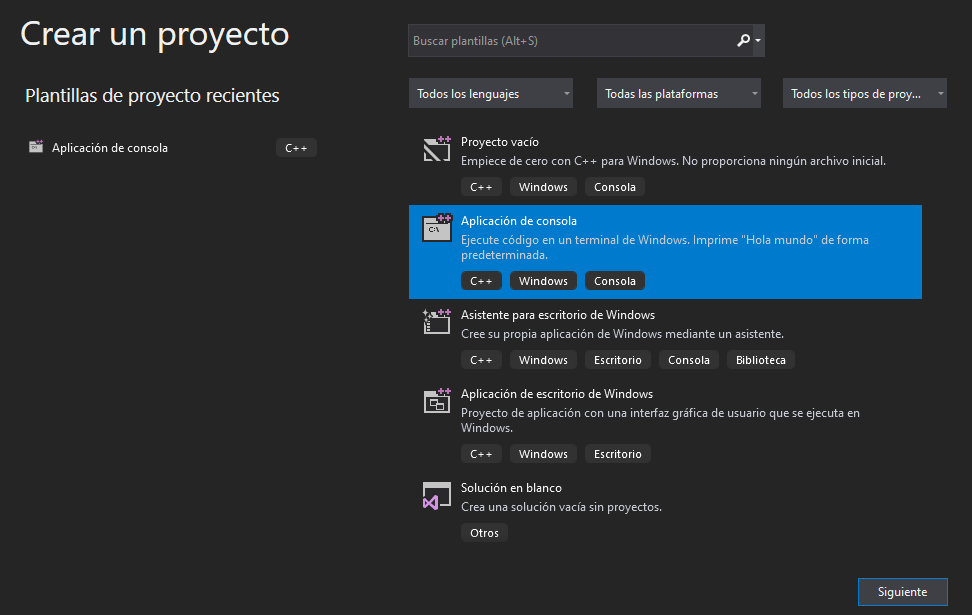
Entonces podríamos pensar en algo como lo mostrado en la siguiente imagen.



## Proceso

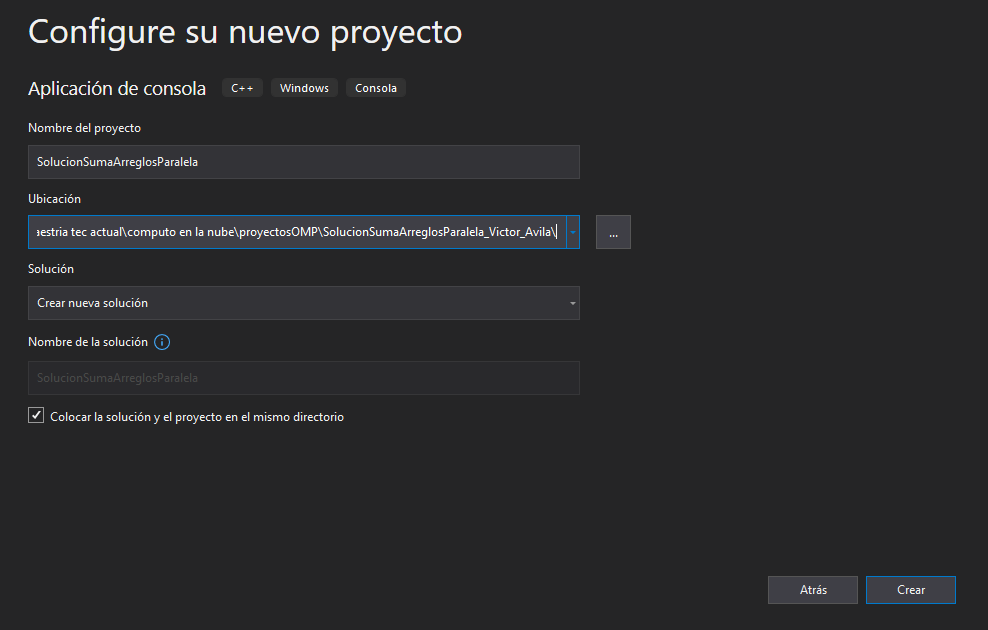
### Crear un proyecto en Visual Studio

Creamos un proyecto nuevo de aplicación en consola dentro de VisualStudio. En mi equipo cuento con la versión 2019.

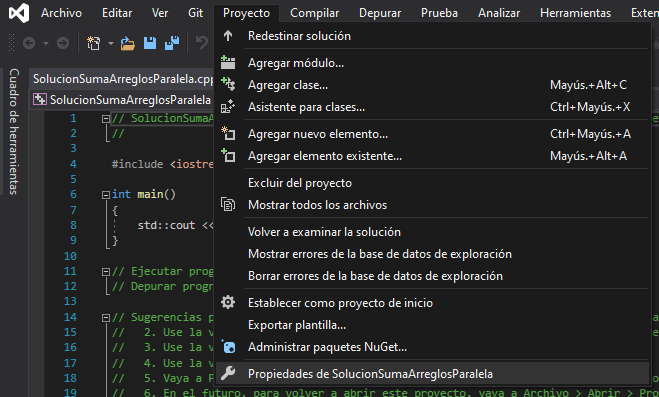


### Configurarlo para que pueda utilizar la librería OpenMP

Configuramos el nombre de la carpeta donde alojaremos los archivos a ejecutar.

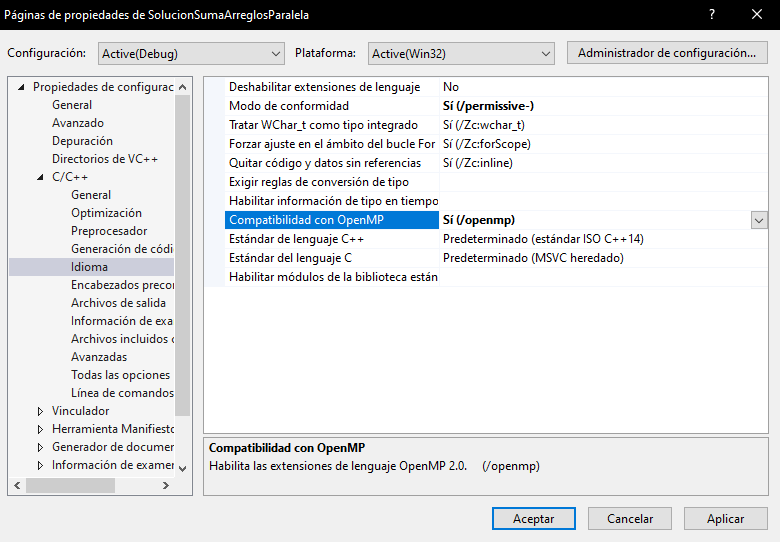


Una vez iniciado el IDE con el proyecto, toca configurar la compatibilidad de este con OpenMP. Ingresamos a la configuración de la siguiente manera:





En el sub-menú idiomas, habilitamos la compatibilidad, y aplicamos los cambios:

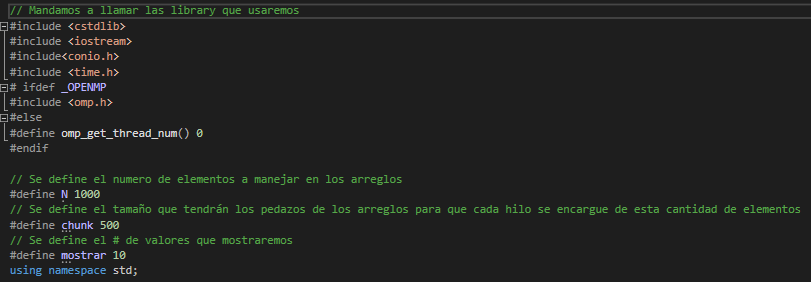


### Crear los arreglos y asignarles valores aleatorios o pedirle los valores al usuario (la cantidad de elementos puede ser de 1000 para ejemplificar el proceso y obtener rápidamente resultados)

Mandamos a llamar las library que usaremos.

Se define:

* el numero de elementos a manejar en los arreglos (N)
* el tamaño que tendrán los pedazos de los arreglos para que cada hilo se encargue de esta cantidad de elementos (chunk)
* el # de valores que mostraremos (mostrar)



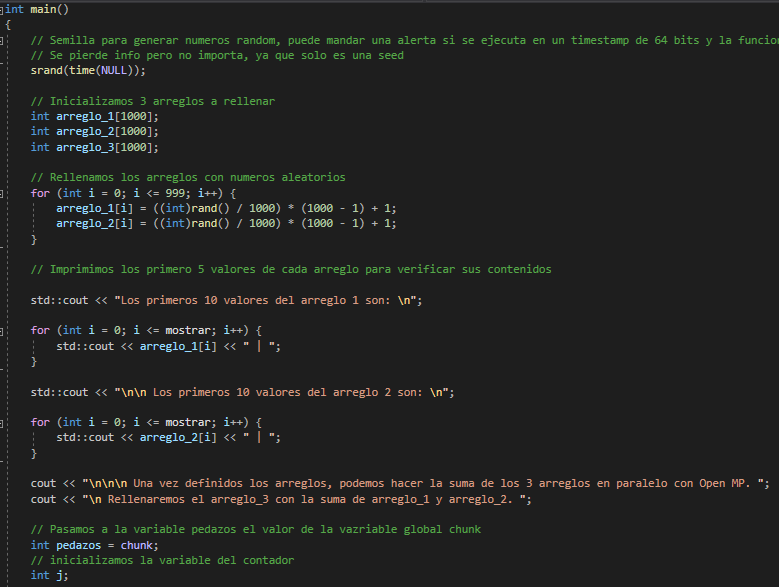
Creamos la semilla para generar números random, puede mandar una alerta si se ejecuta en un timestamp de 64 bits y la función requiere 34; Se pierde info pero no importa, ya que solo es una seed.

Inicializamos 3 arreglos a rellenar (arreglo\_1, arreglo\_2, arreglo\_3)

Rellenamos los arreglos con números aleatorios usando ciclos for ejecutando la función rand() en cada espacio de los arreglos con un máximo de 1000 valores.

Imprimimos los primeros 5 valores de cada arreglo para verificar sus contenidos con bucles for para recorrer los arreglos.

Pasamos a la variable pedazos el valor de la variable global chunk a la variable pedazos e inicializamos la variable del contador local de los hilos (threads).

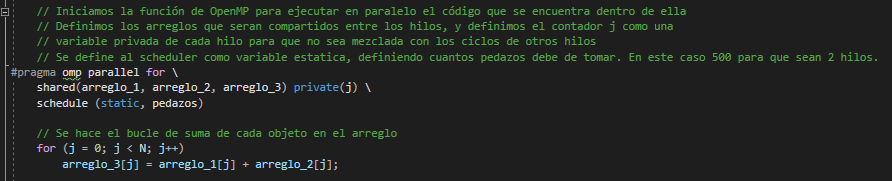


### Realizar un for paralelo a través de las funciones de la librería OpenMP

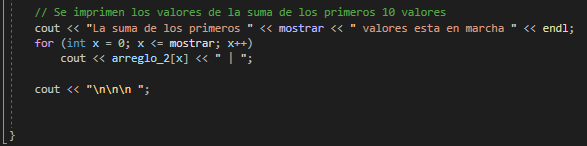
### Iniciamos la función de OpenMP para ejecutar en paralelo el código que se encuentra dentro de ella.

### Definimos los arreglos que serán compartidos entre los hilos, y definimos el contador j como una variable privada de cada hilo para que no sea mezclada con los ciclos de otros hilos.

Se define al scheduler como variable estática, definiendo cuantos pedazos debe de tomar. En este caso 500 para que sean 2 hilos.

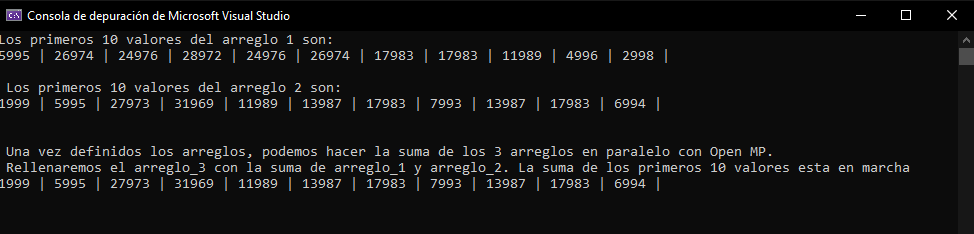


### Imprimir los arreglos o parte de ellos para comprobar que el arreglo resultante contiene la suma de los dos arreglos

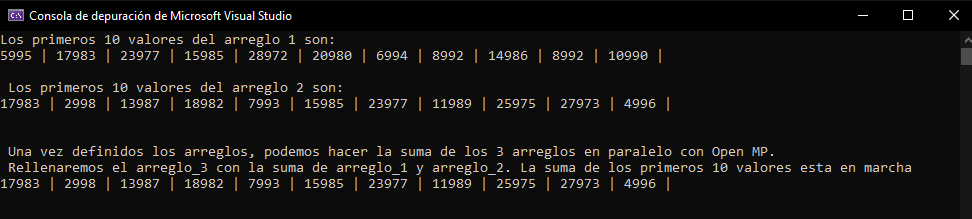


## Resultados

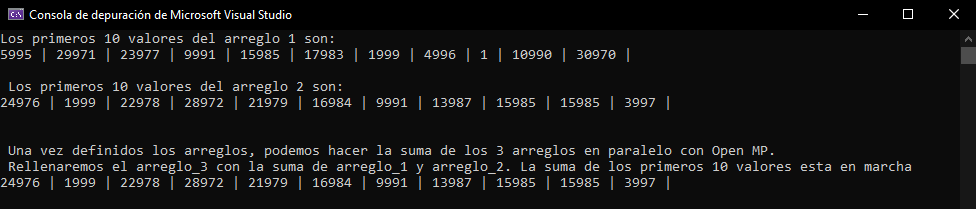
### Ejecución 1



### Ejecución 2



### Ejecución 3



## Reflexión

El uso de las bibliotecas de OpenMP son sencillas cuando se usa la guía Creada por el profesor. Intenté hacerlo de manera independiente, pero me encontré con que la documentación oficial está un poco pesada, ya que te muestran como modificar paso a paso todas las variables, funciones, y definiciones del uso de la biblioteca. Es una gran herramienta para hacer uso de la computación en paralelo, dado que, como vimos en uno de los videos de bibliografía: “La programación de tareas en paralelo esta para quedarse, y se debe hacer a nivel código”. Lo cual me ha dejado con la duda: Si esto es realmente cierto, ¿Dónde queda la implementación de hilos que el ambiente de Hadoop ofrece con su uso de colas, hilos y clusters en un servidor? ¿Requiere de la programación a mano de cada tarea? ¿O por ser un software altamente especializado en la ejecución de ETL y motores tiene cierto límite en la especialización de cada ejecución en paralelo?

También me encontré con el uso de un lenguaje de programación que no tocaba en un buen rato, por lo que tuve que tomar un rato considerable en volverme a familiarizar con los IDE y el lenguaje C++, junto con su esquema de programación orientada a objetos (OOP).